OMG CORBAmed DTF

Resource Access Decision(RAD)

***Errata* of Revised Submission (corbamed/99-03-02)**

2AB, INC.
Baptist Health Systems of South Florida
CareFlow|Net, Inc.
IBM

OMG TC Document corbamed/99-04-xx
26 April 1999

© Copyright 1999 by 2*A*B, Inc.

© Copyright 1999 by Baptist Health Systems of South Florida

© Copyright 1999 by CareFlow|Net, Inc.

© Copyright 1999 by IBM

The submitting companies listed above have all contributed to this document. These companies recognize that this errata is the joint intellectual property of all the submitters, and may be used by any of them in the future, regardless of whether they ultimately participate in a final joint submission.

The companies listed above hereby grant a royalty-free license to the Object Management Group, Inc. (OMG) for worldwide distribution of this document or any derivative works thereof, so long as the OMG reproduces the copyright notices and the below paragraphs on all distributed copies.

The material in this document is submitted to the OMG for evaluation. Submission of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, THE COMPANIES LISTED ABOVE MAKE NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. The companies listed above shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. The information contained in this document is subject to change without notice.

This document contains information which is protected by copyright. All Rights Reserved. Except as otherwise provided herein, no part of this work may be reproduced or used in any form or by any means graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems without the permission of one of the copyright owners. All copies of this document must include the copyright and other information contained on this page.

The copyright owners grant member companies of the OMG permission to make a limited number of copies of this document (up to fifty copies) for their internal use as part of the OMG evaluation process.

RESTRICTED RIGHTS LEGEND. Use, duplication, or disclosure by government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Right in Technical Data and Computer Software Clause at DFARS 252.227.7013.

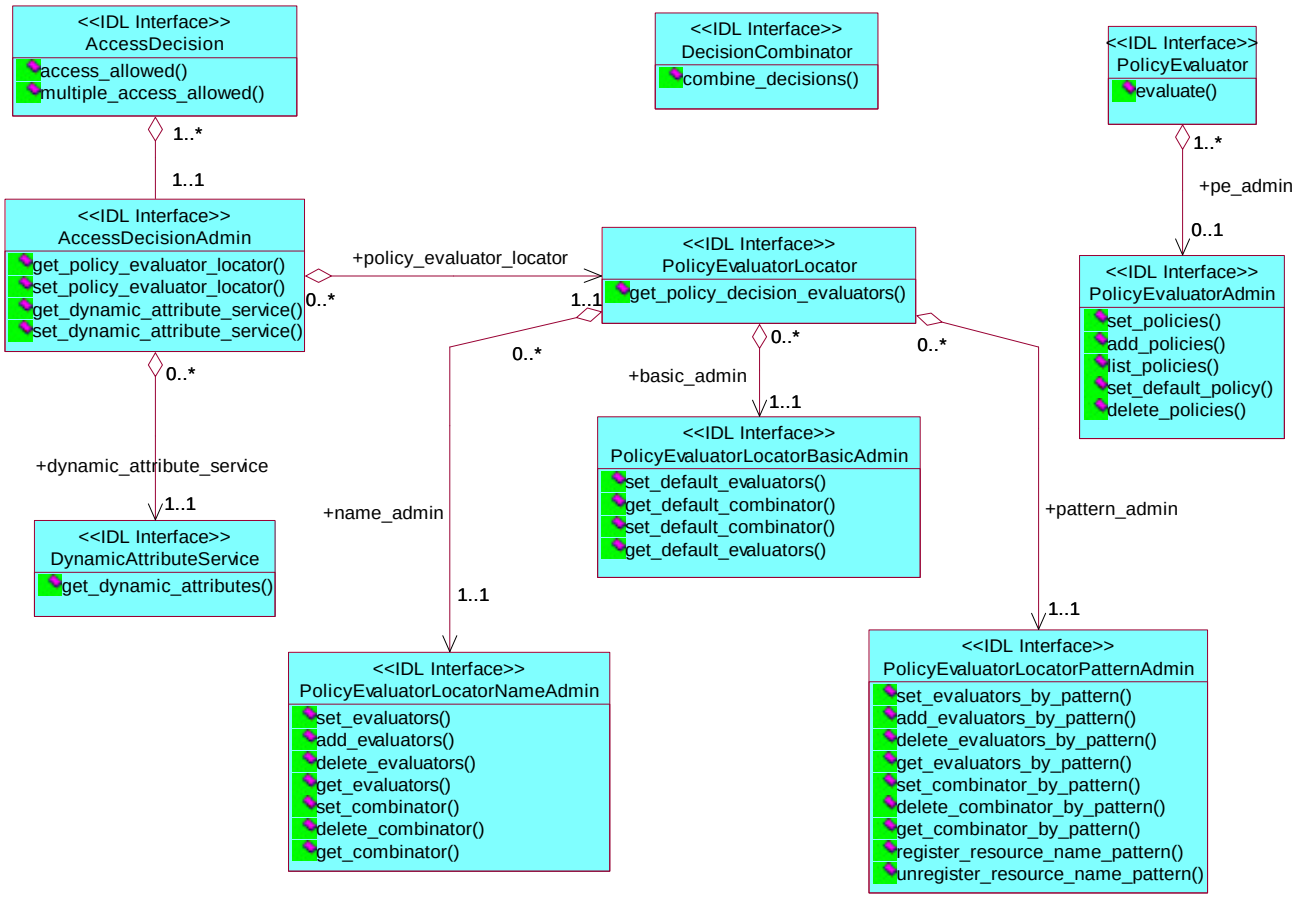
CORBA, OMG and Object Request Broker are trademarks of Object Management Group.

Changes

1. Miscellaneous typos and wording errors were corrected.

2. **Replacing PolicyEvalutatorLocatorAdmin interface with PolicyEvalutatorLocatorBasicAdmin PolicyEvalutatorLocatorNameAdmin, and PolicyEvalutatorLocatorPAtternAdmin**

    - Computational model diagram on page 20 was replaced with the following diagram:



- The second bullet item on page 20 was replaced with the following text:

    - "Administrative interfaces: AccessDecisionAdmin, PolicyEvaluatorAdmin, and PolicyEvaluatorLocatorBasicAdmin, PolicyEvaluatorLocatorNameAdmin, PolicyEvaluatorLocatorPatternAdmin."

- The paragraph on page 35 beginning with words "**`readonly attribute PolicyEvaluatorAdmin`**" was replaced with the following text:

    `"readonly attribute PolicyEvaluatorLocatorBasicAdmin basic_admin`

    The PolicyEvaluatorLocator's basic administrative interface can be obtained via this attribute.

    `readonly attribute PolicyEvaluatorLocatorNameAdmin name_admin`

    The interface for administrating associations between resource names and policy evaluators as well as between resource names and decision combinators can be obtained via this attribute.

readonly attribute PolicyEvaluatorLocatorPatternAdmin pattern_admin

The interface for administrating associations between resource name patterns and policy evaluators as well as between resource name patterns and decision combinators can be obtained via this attribute. If an implementation of a policy evaluator locator does not implement support for resource name patterns this attribute must be *null*."

- IDL code fragment on page 35 was replaced with the following:

```
//*****************************************************
//     interface PolicyEvaluatorLocator
//*****************************************************

interface PolicyEvaluatorLocator {

    readonly attribute PolicyEvaluatorLocatorBasicAdmin basic_admin;

    readonly attribute PolicyEvaluatorLocatorNameAdmin name_admin;

    readonly attribute PolicyEvaluatorLocatorPatternAdmin pattern_admin;

    PolicyDecisionEvaluators get_policy_decision_evaluators(
        in  ResourceName    resource_name
    )
    raises (ComponentError);

};
```

- Section 3.8 "*PolicyEvaluatorLocatorAdmin interface*" was deleted.
- After section 3.7 "*AccessDecisionAdmin interface*" the following new sections were inserted:
  "

3.8 PolicyEvaluatorLocatorBasicAdmin interface

```
//*****************************************************
//     interface PolicyEvaluatorLocatorBasicAdmin
//*****************************************************

interface PolicyEvaluatorLocatorBasicAdmin {

    PolicyEvaluatorList set_default_evaluators(
        in  PolicyEvaluatorList policy_evaluator_list
    )
    raises (DuplicateEvaluatorName, InvalidPolicyEvaluatorList);

   PolicyEvaluatorList get_default_evaluators();

    DecisionCombinator get_default_combinator ();

    void set_default_combinator(
        in  DecisionCombinator decision_combinator
    );
};
```

The PolicyEvaluatorLocatorBasicAdmin object is used to administrate default associations between PolicyEvaluators and ResourceNames as well as default associations between DecisionCombinators and ResourceNames.

`set_default_evaluators()`

The list of PolicyEvaluators provided is set as the default evaluators for any ResourceName for which PolicyEvaluators have not been explicitly assigned. Default evaluators are overridden by the add_evaluators() or replace_evaluators() methods. The default evaluators will be returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method when no PolicyEvaluators have been explicitly assigned for a ResourceName.

Preconditions

No preconditions.

Postconditions

1. default_evaluators == "policy_evaluator_list"

`get_default_evaluators()`

The default set of policy evaluators provided is returned.

Preconditions

No preconditions.

Postconditions

1. return == default_evaluators.

`get_default_combinator()`

The DecisionCombinator provided is returned.

Preconditions

No preconditions.

Postconditions

2. return == default_combinator.

`set_default_combinator()`

The DecisionCombinator provided is set as a default. This combinator is now the combinator used when a DecisionCombinator has not been explicitly specified for a secured resource. This combinator will be returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method for these resources.

Preconditions

No preconditions.

Postconditions

1. default_combinator == "decision_combinator".

3.9 PolicyEvaluatorLocatorNameAdmin interface

```
//*****************************************************
//     interface PolicyEvaluatorLocatorNameAdmin
//*****************************************************

interface PolicyEvaluatorLocatorNameAdmin {

    PolicyEvaluatorList get_evaluators(
        in  ResourceName resource_name
    )
    raises (InvalidResourceName);

    void set_evaluators (
        in  PolicyEvaluatorList policy_evaluator_list,
        in      ResourceName resource_name
    )
    raises (InvalidPolicyEvaluatorList,
            InvalidResourceName,
            DuplicateEvaluatorName);

    void add_evaluators (
        in  PolicyEvaluatorList policy_evaluator_list,
        in  ResourceNamePattern resource_name
    )
    raises (InvalidResourceName,
            InvalidPolicyEvaluatorList,
            DuplicateEvaluatorName);

    void delete_evaluators (
        in  PolicyEvaluatorList policy_evaluator_list,
        in   ResourceNamePattern resource_name
    )
    raises (InvalidResourceName,
            InvalidPolicyEvaluatorList,
            DuplicateEvaluatorName);

    DecisionCombinator get_combinator (
        in  ResourceNamePattern resource_name
    )
    raises (InvalidResourceName);

    void set_combinator (
        in  DecisionCombinator decision_combinator,
        in  ResourceNamePattern resource_name
    )
    raises (InvalidResourceName);

    void delete_combinator (
        in  ResourceNamePattern resource_name
    )
    raises (InvalidResourceName);
};
```

The PolicyEvaluatorLocatorNameAdmin object is used to associate PolicyEvaluators with a ResourceName. The object is also used to associate the appropriate DecisionCombinator with a ResourceName. This submission provides a framework for the support of one or more policy evaluators for a single resource.

`get_evaluators()`

The list of PolicyEvaluators associated with the ResourceNamePattern is returned.

Preconditions

No preconditions.

Postconditions

1. return == "resource_name".registered_ evalautor_list

`set_evaluators()`

A list of PolicyEvaluators is assigned for the named resource. If the resource had existing PolicyEvaluators assigned, they are removed and the entire list is replaced with the ones provided in this method. The replacement of evaluators for a resource which previously had none results in the added list of evaluators being the only evaluators consulted on an access decision (system default evaluators are no longer consulted unless a system default evaluator is a member of the replacement list).

These evaluators will be the PolicyEvaluators returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method.

Preconditions

No preconditions.

Postconditions

1. “resource_name”.registered_evaluator_list == policy_evaluator_list

`add_evaluators()`

A list of PolicyEvaluators is added to the list of evaluators for the named resource. These evaluators will be in the list of PolicyEvaluators returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method. The addition of evaluators to a ResourceName which previously had none results in the added list of evaluators being the only evaluators consulted on an access decision (system default evaluators are no longer consulted unless a system default evaluator is a member of the added list).

Preconditions

No preconditions.

Postconditions

1. “resource_name”.registered_evaluator_list == union (policy_evaluator_list, “resource_name”.registered_evaluator_list)

`delete_evaluators()`

The list of PolicyEvaluators is removed from the list of evaluators for the named resource. These evaluators will not be in the list of PolicyEvaluators returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method.

Preconditions

No preconditions.

Postconditions

1. for the "resource_name" : “resource_name”.registered_evaluator_list = “resource_name”.registered_evaluators - "policy_evaluator_list"

`get_combinator()`

The DecisionCombinator specified for the named resource is returned. If a combinator has not been specified for the ResourceName provided, the return will be nil (it will not return the default combinator).

Preconditions

No preconditions.

Postconditions

1. return == "resource_name".registered_ decision_combinator

`set_combinator()`

A DecisionCombinator is specified for the named resource. This combinator will be returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method. The DecisionCombinator provided replaces any previous combinator specified for the secured resource.

Preconditions

No preconditions.

Postconditions

1. "resource_name".registered_ decision_combinator == “decision_combinator”

`delete_combinator()`

The DecisionCombinator for the ResourceName is removed. The default combinator will now be returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method.

Preconditions

No preconditions.

Postconditions

1. Resource names matching only "resource_name" will be associated with the default combinator.

3.10 PolicyEvaluatorLocatorPatternAdmin interface

```
//*****************************************************
//     interface PolicyEvaluatorLocatorPatternAdmin
//*****************************************************

interface PolicyEvaluatorLocatorPatternAdmin {

    void register_resource_name_pattern(
        in  ResourceNamePattern pattern
    )
    raises (InvalidResourceNamePattern,
            PatternDuplicate,
            PatternConflict);

    void unregister_resource_name_pattern(
        in  ResourceNamePattern pattern
    )
    raises (InvalidResourceNamePattern,
```

```
            PatternNotRegistered,
            PatternInUse);

    PolicyEvaluatorList get_evaluators_by_pattern (
        in  ResourceNamePattern pattern
    )
    raises (InvalidResourceNamePattern,
            PatternNotRegistered);

    void set_evaluators_by_pattern (
        in  PolicyEvaluatorList policy_evaluator_list,
        in      ResourceNamePattern pattern
    )
    raises (InputFormatError,
            PatternNotRegistered,
            DuplicateEvaluatorName);

    void add_evaluators_by_pattern (
        in  PolicyEvaluatorList policy_evaluator_list,
        in  ResourceNamePattern pattern
    )
    raises (InvalidResourceNamePattern,
            PatternNotRegistered,
            InvalidPolicyEvaluatorList,
            DuplicateEvaluatorName);

    void delete_evaluators_by_pattern (
        in  PolicyEvaluatorList policy_evaluator_list,
        in   ResourceNamePattern pattern
    )
    raises (InvalidResourceNamePattern,
            PatternNotRegistered,
            InvalidPolicyEvaluatorList,
            DuplicateEvaluatorName);

    DecisionCombinator get_combinator_by_pattern (
        in  ResourceNamePattern pattern
    )
    raises (InvalidResourceNamePattern,
            PatternNotRegistered);

    void set_combinator_by_pattern (
        in  DecisionCombinator decision_combinator,
        in  ResourceNamePattern pattern
    )
    raises (InvalidResourceNamePattern,
            PatternNotRegistered);

    void delete_combinator_by_pattern (
        in  ResourceNamePattern pattern
    )
    raises (InvalidResourceNamePattern,
            PatternNotRegistered);
};
```

The PolicyEvaluatorLocatorPatternAdmin object is used to associate PolicyEvaluators with a ResourceNamePatterns. The object is also used to associate the appropriate DecisionCombinator with the ResourceNamePattern. This submission provides a framework for the support of one or more policy evaluators for a single resource pattern.

Patterns are used to group resource names without requiring the PolicyEvaluatorLocator administrator to enumerate all the resources names individually; this is accomplished by associating lists of PolicyEvaluator objects with ResourceNamePatterns, and checking whether a supplied resource name matches any of the Patterns with which it has associated PolicyEvaluators. This section describes how RAD objects decide whether a Pattern matches a resource name. Throughout the section, we use the shorthand phrase "exactly matches" to mean "is exactly the same string as". Patterns have a specific format:

- A Pattern must include a ResourceNamingAuthority.
- A Pattern must include a list of ResourceNameComponent strings.
- Each ResourceNameComponent consists of a name_string and a value_string.

Two kinds of ResourceNameComponents can occur in a pattern.

The first kind is a component value pattern. It has the form:

- name_string is a string
- value_string is a regular expression

A resource name component matches a component value pattern only if its

- name_string exactly matches the pattern's name_string and its value_string matches the component value pattern's value_string regular expression.

The second kind of ResourceNameComponent which can occur in a pattern is a component wildcard pattern:

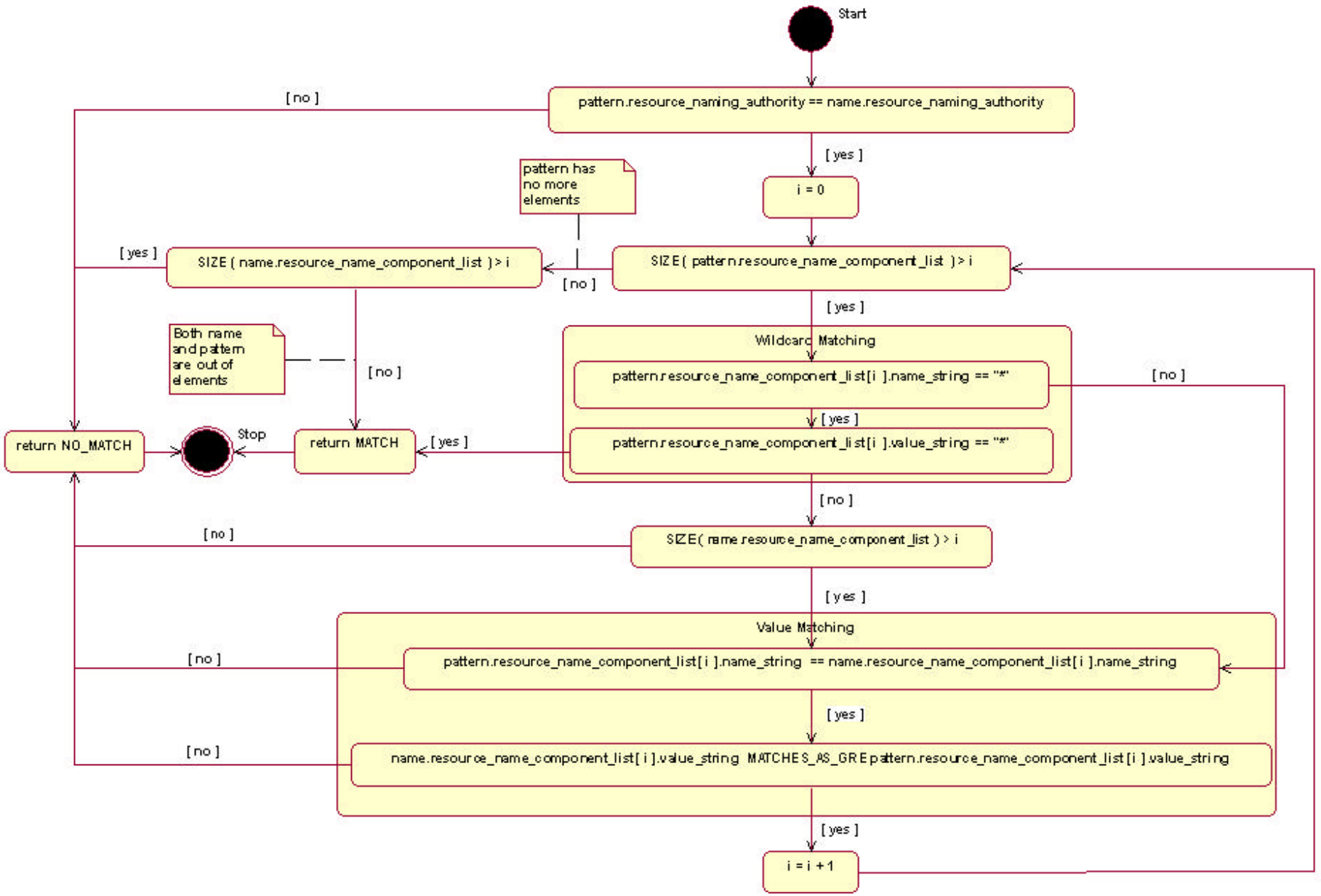
- name_string is "*"
- value_string is "*"

Every component of a resource name matches a component wildcard pattern.

A resource name matches a pattern if and only if the algorithm shown in the figure below returns MATCH.

The algorithm has two inputs: resource name (“name”) and resource name pattern (“pattern”). It also assumes availability of two functions:

- SIZE – returns number of elements in a sequence,
- MATCHES_AS_GRE – returns “yes” if the string provided by first argument matches another string provided by second argument, where the second string is interpreted according to regular expression syntax specified in the definition of ResourceNamePattern type on Page 27, otherwise “no”.


Start
pattern.resource_naming_authority == name.resource_naming_authority
[no]
[yes]
i = 0
pattern has no more elements
SIZE (pattern.resource_name_component_list) > i
[no]
[yes]
SIZE (name.resource_name_component_list) > i
[yes]
Wildcard Matching
Both name and pattern are out of elements
[no]
pattern.resource_name_component_list[i].name_string == "*"
[no]
[yes]
pattern.resource_name_component_list[i].value_string == "*"
return NO_MATCH
Stop
return MATCH
[yes]
[no]
SIZE(name.resource_name_component_list) > i
[no]
[yes]
Value Matching
pattern.resource_name_component_list[i].name_string == name.resource_name_component_list[i].name_string
[no]
[yes]
name.resource_name_component_list[i].value_string MATCHES_AS_GRE pattern.resource_name_component_list[i].value_string
[no]
[yes]
i = i + 1

`register_resource_name_pattern()`

Before a ResourceNamePattern can be used in the administrative interfaces, it must be registered. This allows the administration of name patterns separately from the administration of the association of patterns to evaluators and combinators. Since a ResourceName is a ResourceNamePattern, ResourceNames must also be registered if these administrative interfaces are administer evaluators and combinators.

Implementations may or may not support overlapping patterns; that is, an implementation may choose to allow registration of two patterns both of which match at least one name, or they may choose not to allow such registrations. An implementation which does not support overlapping patterns shall raise the PatternConflict exception when this method is used to register a pattern which overlaps with another previously registered pattern. Implementors should document whether their implementations support overlapping patterns or not.

Preconditions

No preconditions.

Postconditions

1. "resource_name_pattern" is registered.

`unregister_resource_name_pattern()`

ResourceNamePatterns may be unregistered. A ResourceNamePattern must not have any evaluators or combinators associated with it when it is unregistered.

Preconditions

No preconditions.

Postconditions

1. "resource_name_pattern" is unregistered.

`get_evaluators_by_pattern ()`

The list of PolicyEvaluators associated with the ResourceNamePattern is returned.

Preconditions

No preconditions.

Postconditions

2. return == "resource_name_pattern".registered_ evalautor_list

`set_evaluators_by_pattern ()`

A list of PolicyEvaluators is assigned for the named resource. If the resource had existing PolicyEvaluators assigned, they are removed and the entire list is replaced with the ones provided in this method. The replacement of evaluators for a resource which previously had none results in the added list of evaluators being the only evaluators consulted on an access decision (system default evaluators are no longer consulted unless a system default evaluator is a member of the replacement list).

These evaluators will be the PolicyEvaluators returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method.

Preconditions

No preconditions.

Postconditions

2. “resource_name_pattern”.registered_evaluator_list == policy_evaluator_list

`add_evaluators_by_pattern ()`

A list of PolicyEvaluators is added to the list of evaluators for the named resource. These evaluators will be in the list of PolicyEvaluators returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method. The addition of evaluators to a ResourceNamePattern which previously had none results in the added list of evaluators being the only evaluators consulted on an access decision (system default evaluators are no longer consulted unless a system default evaluator is a member of the added list).

Preconditions

No preconditions.

Postconditions

2. “resource_name_pattern”.registered_evaluator_list == union (policy_evaluator_list, “resource_name_pattern”.registered_evaluator_list)

`delete_evaluators_by_pattern ()`

The list of PolicyEvaluators is removed from the list of evaluators for the named resource. These evaluators will not be in the list of PolicyEvaluators returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method.

Preconditions

No preconditions.

Postconditions

2. for the "resource_name_pattern" : “resource_name_ pattern”.registered_evaluator_list = “resource_name_ pattern”.registered_evaluators - "policy_evaluator_list"

`get_combinator_by_pattern ()`

The DecisionCombinator specified for the named resource is returned. If a combinator has not been specified for the ResourceNamePattern provided, the return will be nil (it will not return the default combinator).

Preconditions

No preconditions.

Postconditions

2. return == "resource_name_pattern".registered_ decision_combinator

`set_combinator_by_pattern ()`

A DecisionCombinator is specified for the named resource. This combinator will be returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method. The DecisionCombinator provided replaces any previous combinator specified for the secured resource.

Preconditions

No preconditions.

Postconditions

2. "resource_name_pattern".registered_ decision_combinator == “decision_combinator”

`delete_combinator_by_pattern ()`

The DecisionCombinator for the ResourceNamePattern is removed. The default combinator will now be returned by the PolicyEvaluatorLocator get_policy_decision_evaluators() method.

Preconditions

No preconditions.

Postconditions

2. Resource names matching only "resource_name_ pattern" will be associated with the efault combinator.”

- IDL code in Appendix 7 “Complete IDL” was updated to reflect that new interfaces: *PolicyEvaluatorLocatorBasicAdmin, PolicyEvaluatorLocatorNameAdmin, and PolicyEvaluatorLocatorPatternAdmin* replace interface *PolicyEvaluatorLocatorAdmin.*

1. Introducing conformance classes

- The contents of Chapter 4 “Conformance Classes” was replaced with the following text:

“There are two conformance classes: “RAD without Patterns” and “RAD with Patterns”.

An implementation of Resource Access Decision (RAD) facility compliant to conformance class “**RAD without Patterns**” must implement all of the interfaces defined in this submission except interface *PolicyEvaluatorLocatorPatternAdmin.* In this case *pattern_admin* data member of *PolicyEvaluatorLocator* interface implementation must have value *null.*

An implementation of Resource Access Decision facility compliant to conformance class **“RAD with Patterns**” must implement all of the interfaces defined in this submission. In this case *pattern_admin* data member of *PolicyEvaluatorLocator* interface implementation must have non *null* value.”